Concepts

Flex Container and Flex Items

- **Flex Container**: The element on which `display: flex;` is applied. This container becomes a flex container, and its direct children become flex items.
- **Flex Items**: The children of the flex container. These items will automatically align themselves according to the flexbox rules.

Creating a Flex Container

To start using Flexbox, you need to define a flex container. Here's how:

```
<div class="container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

.container {
  display: flex; /* This makes the container a flex container */
  border: 2px solid black;
}

.item {
  background-color: lightblue;
  padding: 10px;
  border: 1px solid blue;
  margin: 5px;
}
```

2. Flexbox Properties

Main Axis and Cross Axis

- **Main Axis**: The primary axis along which flex items are laid out. By default, this is horizontal (left to right).
- **Cross Axis**: The axis perpendicular to the main axis (vertical by default).

Direction of Flex Items

You can change the direction of the flex items using `flex-direction`.

```
.container {
  display: flex;
  flex-direction: row; /* Default: items arranged horizontally */
}
```

Other values for `flex-direction`:

- `row`: Items arranged horizontally (default).
- `row-reverse`: Items arranged horizontally but in reverse order.
- `column`: Items arranged vertically.
- `column-reverse`: Items arranged vertically but in reverse order.

Aligning Items

- **Justify Content (Main Axis)** Controls alignment along the main axis.

```
.container {
  display: flex;
  justify-content: center; /* Align items to the center horizontally
*/
}
```

Other values for `justify-content`:

- `flex-start`: Items align to the start of the container (default).
- `flex-end`: Items align to the end of the container.
- `center`: Items align at the center.
- `space-between`: Items are evenly distributed with the first item at the start and the last item at the end.
- `space-around`: Items are evenly distributed with equal space around them.
- **Align Items (Cross Axis)** Controls alignment along the cross axis.

```
.container {
  display: flex;
  align-items: center; /* Align items to the center vertically */
}
```

Other values for `align-items`:

- `stretch`: Items stretch to fill the container (default).
- `flex-start`: Items align to the start of the container.
- `flex-end`: Items align to the end of the container.
- `center`: Items align at the center.
- `baseline`: Items align to the baseline of their content.

3. Advanced Flexbox Properties

Flex Wrap

By default, flex items try to fit into one line. If you want them to wrap to multiple lines, use `flex-wrap`.

```
.container {
  display: flex;
  flex-wrap: wrap; /* Items will wrap to the next line if needed */
}
```

Other values for `flex-wrap`:

- `nowrap`: All items will be on one line (default).
- `wrap`: Items wrap onto multiple lines.
- `wrap-reverse`: Items wrap onto multiple lines in reverse order.

Align Content

Controls space between lines when there is extra space in the cross axis.

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: center; /* Align lines to the center vertically */
}
```

Other values for `align-content`:

- `flex-start`: Lines packed to the start of the container.
- `flex-end`: Lines packed to the end of the container.

- `center`: Lines packed to the center.
- `space-between`: Lines evenly distributed with the first line at the start and the last line at the end.
- `space-around`: Lines evenly distributed with equal space around them.
- `stretch`: Lines stretch to fill the container (default).

Order

You can change the order of flex items using the `order` property.

```
.item {
  order: 2; /* Default is 0; a higher number means the item appears
later */
}
```

- **flex-grow**: Defines the ability of a flex item to grow if necessary.

```
.item {
  flex-grow: 1; /* All items will grow equally */
}
```

4. Complete Example

Here's a complete example that combines many of these properties:

```
<div class="container">
  <div class="item" style="flex-grow: 2;">Item 1</div>
  <div class="item" style="flex-grow: 1;">Item 2</div>
  <div class="item" style="flex-grow: 1;">Item 3</div>
</div>
```

css

```
.container {
  display: flex;
  justify-content: space-around;
  align-items: center;
  flex-wrap: wrap;
  height: 200px;
  border: 2px solid black;
}

.item {
  background-color: lightblue;
  padding: 10px;
  border: 1px solid blue;
  margin: 5px;
}
```

What is a Media Query?

A media query consists of a media type and one or more expressions that check for the conditions of particular media features (like screen width). If the condition(s) in the query are true, the styles inside the media query will be applied.

Syntax of a Media Query

The basic syntax looks like this:

```
@media (condition) {
  /* CSS rules */
}
```

2. Media Types

- **all**: Suitable for all devices.
- **screen**: Used for computer screens, tablets, smartphones, etc.

Example:

```
@media screen {
  body {
    background-color: lightblue;
  }
}
```

This query applies the styles only to devices with a screen.

3. Media Features

Media features are the conditions you can check for within your media queries. The most common one is `width`, but there are many others.

Width and Height

- **min-width**: Sets styles if the viewport width is greater than or equal to a certain value.
- **max-width**: Sets styles if the viewport width is less than or equal to a certain value.
- **min-height** and **max-height**: Similar to `min-width` and `max-width` but for height.

Example:

```
@media (min-width: 768px) {
  body {
    background-color: lightgreen;
  }
}
```

This changes the background color to light green if the viewport width is at least 768 pixels.

Orientation

- **orientation: landscape**: Applies styles if the device is in landscape mode.
- **orientation: portrait**: Applies styles if the device is in portrait mode.

Example:

```
@media (orientation: landscape) {
  body {
    background-color: lightcoral;
  }
}
```

This changes the background color to light coral if the device is in landscape orientation.

4. Combining Multiple Conditions

You can combine multiple conditions using logical operators like `and`, `or`, and `not`.

Using `and`

```
@media (min-width: 768px) and (max-width: 1024px) {
  body {
    background-color: lightpink;
  }
}
```

This changes the background color to light pink if the viewport width is between 768px and 1024px.

Using `or`

```
@media (max-width: 600px), (orientation: portrait) {
  body {
    background-color: lightgray;
  }
}
```

This applies the styles if the viewport width is 600px or less, **or** if the device is in portrait mode.

Using `not`

```
@media not all and (min-width: 768px) {
  body {
    background-color: lightyellow;
  }
}
```

This changes the background color to light yellow if the viewport width is **not** at least 768px.

What is a CSS Animation?

CSS animations enable you to animate transitions between different states of an element. Unlike transitions, which happen between two states (e.g., from one color to another), animations can define multiple states.

2. Keyframes

The `@keyframes` rule is used to create animations. It defines the start, end, and intermediate points of the animation.

Syntax:

```
@keyframes animationName {

  0% {

    /* Start state */

  }

  50% {

    /* Mid state */

  }

  100% {

    /* End state */

  }

}
```

Example:

```
@keyframes moveRight {

  0% {

    transform: translateX(0);

  }

  100% {

    transform: translateX(100px);

  }

}
```

In this example, the `moveRight` animation starts with the element in its original position and moves it 100px to the right.

3. Applying Animations

To apply an animation, you use the `animation` property. This property can include several values:

- **animation-name**: The name of the `@keyframes` animation.
- **animation-duration**: How long the animation takes to complete one cycle.
- **animation-timing-function**: The speed curve of the animation (e.g., `ease`, `linear`, `ease-in`, `ease-out`).
- **animation-delay**: When the animation starts.
- **animation-iteration-count**: How many times the animation should run (e.g., `infinite` for endless looping).
- **animation-direction**: The direction of the animation (e.g., `normal`, `reverse`, `alternate`).

Example:

```
<div class="box"></div>



<style>

  .box {

    width: 100px;

    height: 100px;

    background-color: lightblue;

    animation: moveRight 2s ease-in-out infinite;

  }



  @keyframes moveRight {

    0% {
      transform: translateX(0);

    }

    100% {

      transform: translateX(100px);

    }

  }

</style>
```

4. Advanced Animation Properties

Animation Timing Functions

- **linear**: The animation proceeds at a constant speed.
- **ease**: The default; starts slow, speeds up, then slows down.
- **ease-in**: Starts slowly.
- **ease-out**: Ends slowly.
- **ease-in-out**: Starts and ends slowly.

```
animation-timing-function: ease-in-out;
```

Animation Delay

You can delay the start of an animation:

```
animation-delay: 1s;
```

Animation Iteration Count

Control how many times an animation repeats:

```
animation-iteration-count: 3; /* Runs 3 times */

animation-iteration-count: infinite; /* Runs forever */
```

Animation Direction

- **normal**: The animation runs forward (default).
- **reverse**: The animation runs backward.
- **alternate**: The animation runs forward first, then backward.
- **alternate-reverse**: The animation runs backward first, then forward.

```
animation-direction: alternate;
```

5. Combining Multiple Animations

You can apply multiple animations to an element by separating them with commas:

```
animation: moveRight 2s ease-in-out, changeColor 2s ease-in-out;
```

6. Animation Example

Let's create an animation that moves a box from left to right, changes its color, and repeats infinitely:

```
<div class="box"></div>



<style>

  .box {

    width: 100px;

    height: 100px;

    background-color: lightblue;

    position: relative;
    animation: moveRight 3s ease-in-out infinite alternate,
changeColor 3s ease-in-out infinite alternate;

  }


  @keyframes moveRight {

    0% {

      left: 0;

    }

    100% {

      left: 300px;

    }

  }


  @keyframes changeColor {

    0% {

      background-color: lightblue;

    }

    100% {

      background-color: lightcoral;

    }

  }

</style>
```

In this example:

- The `box` moves from left to right over 3 seconds, then back to the left (because of `alternate`).
- The color changes from `lightblue` to `lightcoral` and back.

7. Shorthand `animation` Property

You can combine all animation properties into a single `animation` shorthand:

```
animation: moveRight 3s ease-in-out 1s infinite alternate;
```

This shorthand includes:

1. `animation-name`
2. `animation-duration`
3. `animation-timing-function`
4. `animation-delay`
5. `animation-iteration-count`
6. `animation-direction`

Translate

- Moves an element from its current position.
- `translateX(n)`: Moves the element horizontally by `n` pixels.
- `translateY(n)`: Moves the element vertically by `n` pixels.
- `translate(n, m)`: Moves the element both horizontally and vertically.

Example:

```
transform: translateX(50px); /* Moves the element 50px to the right */
```

Rotate

- Rotates an element around a fixed point (usually its center).
- `rotate(angle)`: Rotates the element by the specified angle (in degrees).

Example:

```css
transform: rotate(45deg); /* Rotates the element by 45 degrees */
```

Scale

- Resizes an element.
- `scaleX(n)`: Scales the element horizontally by `n`.
- `scaleY(n)`: Scales the element vertically by `n`.
- `scale(n)`: Scales the element uniformly in both directions.

Example:

```css
transform: scale(1.5); /* Increases the size by 1.5 times */
```

Skew

- Distorts an element by a given angle along the X or Y axis.
- `skewX(angle)`: Skews the element along the X-axis.
- `skewY(angle)`: Skews the element along the Y-axis.

Example:

```css
transform: skewX(30deg); /* Skews the element 30 degrees along the
X-axis */
```

2. Combining Multiple Transforms

You can combine multiple transform functions by chaining them together:

```css
transform: translateX(50px) rotate(45deg) scale(1.5);
```

3. Applying Transforms in Animations

You can animate transformations using keyframes. Here’s how:

Example: Moving and Rotating a Box

Let's create an animation that moves and rotates a box:

```html
<div class="box"></div>
```

```
<style>
  .box {
    width: 100px;
    height: 100px;
    background-color: lightblue;
    position: relative;
    animation: moveRotate 4s infinite;
  }

  @keyframes moveRotate {
    0% {
      transform: translateX(0) rotate(0deg);
    }
    50% {
      transform: translateX(200px) rotate(180deg);
    }
    100% {
      transform: translateX(0) rotate(360deg);
    }
  }
</style>
```

Explanation:

- **Keyframes**: The animation progresses through three states—starting at `0%`, halfway at `50%`, and ending at `100%`.
- **Transformations**:
  - At `0%`: The box is in its original position and orientation.
  - At `50%`: The box moves 200px to the right and rotates 180 degrees.
  - At `100%`: The box returns to its original position and completes a full 360-degree rotation.

Rotate in 3D

You can rotate an element in 3D space using `rotateX`, `rotateY`, and `rotateZ`:

```
transform: rotateX(30deg) rotateY(30deg) rotateZ(30deg);
```

Example: 3D Rotation

```
<div class="box"></div>

<style>
  .box {
    width: 100px;
    height: 100px;
    background-color: lightblue;
    position: relative;
    transform-style: preserve-3d;
    animation: rotate3D 4s infinite;
  }

  @keyframes rotate3D {
    0% {
      transform: rotateX(0) rotateY(0);
    }
    50% {
      transform: rotateX(180deg) rotateY(180deg);
    }
    100% {
      transform: rotateX(360deg) rotateY(360deg);
    }
  }
</style>
```

++++++++++++